LISTING OF CLAIMS

1. (Original) A symmetric Finite Impulse Response (FIR) filter providing linear scalability and implementation without the need for delay lines, comprising:

a multiprocessor architecture including a plurality of ALUs (Arithmetic and Logic Unit), Multipliers units, Data cache, and Load/Store units sharing a common Instruction cache, and multi-port memory, and

an assigning means for assigning to each available processing unit the computation of specified unique partial product terms and the accumulation of each computed partial product on specified output sample values.

2. (Original) The FIR filter as in claim 1, wherein the assigning means is a preprocess that is used to design the implementation of the FIR filter based on the filter specifications.

3. (Original) A method for implementing an improved symmetric Finite Impulse Response (FIR) filter providing linear scalability using a multiprocessing architecture platform without the need for delay lines, comprising the steps of:

assigning to each available processing unit the computation of specified unique partial product terms; and

accumulating each computed partial product on specified output sample values.

4. (Currently Amended) The ~~[[A]]~~ method wherein the multiprocessing architecture comprises ~~for finite impulse response filtering using~~ a 2 parallel-processor architecture, the method comprising:

initializing a loop-index by zero;

loading an input sample for a first processor #~~1~~;

loading a coefficient for the first processor #~~1~~;

multiplying the input sample and coefficient in the first processor #~~1~~;

updating a current output in the first processor #~~1~~;

updating a partial output, if required, in the first processor #~~1~~;

loading an input sample for a second processor #~~2~~;

loading a coefficient for the second processor #~~2~~;

multiplying the input sample and coefficient in the second processor #~~2~~;

updating a current output in the second processor #~~2~~;

updating a partial output, if required, in the second processor #~~2~~;

increment the loop-index by 2; and

stopping if the loop-index equals end, else, returning to loading the input sample for the first processor #~~1~~ and repeating.

5. (Currently Amended) The ~~[[A]]~~ method ~~for symmetric FIR filter computation~~, comprising:

Initializing a loop index by zero;

loading an input sample and coefficients;

multiplying the input sample and coefficient;

updating a current output;

updating a partial output, if required ~~(see Note below)~~;

incrementing the loop index by a number of processors ~~processor~~; and

if loop termination is condition satisfied then:

stopping; else

returning to loading an input sample and coefficients and repeating.

6. (Currently Amended) The ~~[[A]]~~ method ~~for symmetric FIR filter computation~~, comprising:

initializing a loop index by zero;

loading an input sample and coefficients;

multiplying the input sample and coefficient;

updating a current output;

incrementing the loop index by a number of processors being used for the computation; and

if a loop termination condition satisfied then:

stopping;

else, returning to loading an input sample and coefficients and repeating.

7. (Original) A finite impulse response (FIR) filter architecture, comprising:

a plurality of processors arranged in a parallel processing architecture wherein the processors share a common instruction cache and memory;

an execution program that assigns to each available processor the computation of certain specified unique partial product terms and causes the accumulation of each computed partial product on specified output sample values.

8. (Original) The architecture of claim 7 wherein the execution program causes partial product terms computed during a given sample to be reused during a later sample.

9. (Original) The architecture of claim 8 wherein the computed partial product terms are stored in the memory after use during the give sample and later retrieved therefrom for use during the later sample.